



Dossier Projet

Création d'un site web

Nom : RAHMANI Mohammad

Site web: creativbox.mrah.fr

Description : Réseau social pour partager des travaux créatifs

TABLE DES MATIÈRES

I.	INTRODUCTION	4
I.1.	Contexte	4
I.2.	Description du site web	4
I.3.	Technologies utilisées	5
I.3.1.	Pour le Front-end	5
I.3.2.	Pour le Back-end et la base de données	5
I.3.3.	Pour le serveur	6
I.3.4.	Les logiciels et les outils	6
II.	LISTE DES COMPÉTENCES DU RÉFÉRENTIEL QUI SONT COUVERTES PAR LE PROJET	7
II.1.	Maquette et diagrammes	7
II.2.	Réaliser une interface statique et adaptable	9
II.3.	Développer une interface web dynamique	10
II.4.	Réaliser une interface utilisateur et administrateur avec une solution de gestion de contenu et de profil	11
II.5.	Créer une base de données	14
II.6.	Développer les composants d'accès aux données	15
II.7.	Développer la partie Back-end d'une application web ou web mobile	17
II.8.	Elaborer et mettre en œuvre des composants de gestion de contenu	19
III.	CAHIER DES CHARGES	22
III.1.	Fonctionnement de l'application	22
III.1.1.	Permissions	22
III.1.2.	Fonctionnalités en fonction des permissions	22
III.1.3.	Courte description de chaque URL	23
III.2.	Arborescence du site	24
IV.	SPÉCIFICATIONS FONCTIONNELLES	25
IV.1.	Introduction	25
IV.2.	Tableau des fonctions	25
V.	SPÉCIFICATIONS TECHNIQUES	28
V.1.	Stockage et version de contrôle	28
V.2.	Workflow	29
V.3.	Les technologies utilisées	29
V.3.1.	Front-end	29

V.3.2.	Back-end	29
V.3.3.	Base de données	30
V.3.4.	Tests unitaires	31
V.3.5.	Gestion de documentations API	31
VI.	RÉALISATION DU PROJET ET DU DÉVELOPPEMENT	32
VI.1.	Introduction	32
VI.2.	L'environnement	32
VI.3.	Pourquoi une formation dans le développement ?	32
VI.4.	Organisation	33
VI.5.	Rythme	33
VI.6.	La phase de développement	33
VI.6.1.	Planification	34
VI.6.2.	Lancement	34
VI.6.3.	Déroulement	34
VI.6.4.	Finalisation	34
VI.6.5.	Les outils	35
VII.	DÉROULEMENT DU PROJET	36
VII.1.	Introduction	36
VII.2.	Déroulement du projet semaine par semaine	36
VIII.	PRÉSENTATION DE LA FONCTIONNALITÉ LA PLUS REPRÉSENTATIVE	40
VIII.1.	Introduction	40
VIII.2.	Interface utilisateur et administrateur	41
VIII.3.	Comment l'utiliser	43
VIII.4.	Comment cela fonctionne	45
VIII.5.	Conclusion	49
IX.	TEST UNITAIRE	50
X.	RECHERCHES EFFECTUÉES EN ANGLAIS	52
XI.	VEILLE EFFECTUÉE SUR LES VULNÉRABILITÉS DE SÉCURITÉ	53
XII.	CONCLUSION	54

I. INTRODUCTION

I.1. Contexte

Au cours des 6 mois de formation intensive de développeur web et web mobile, au centre ARINFO d'Allonnes, j'ai acquis beaucoup de compétences. Je les ai complétés avec les bonnes pratiques pour pouvoir créer mon projet.

Pendant cette formation, j'ai compris que l'enseignement se complète avec la pratique. Dans le cas contraire, il ne sera jamais possible de progresser.

Malgré les difficultés que j'ai pu rencontrer avec mon niveau de français, je me suis senti très bien intégré au sein de mon groupe. Il y avait une très bonne entente et tout le monde s'entraidaient.

I.2. Description du site web

Les artistes et les créateurs sont principalement actifs sur les réseaux sociaux comme Instagram et Facebook, mais leurs activités ne sont pas référencées sur les moteurs de recherche, comme Google et Bing.

Le site Créativ'Box est conçu avec l'idée d'établir cette connexion.

Il permet d'exprimer leurs activités. Les internautes peuvent accéder à leurs travaux plus rapidement et accéder directement à leurs réseaux sociaux.

Pour cette raison, j'ai créé ce site pour que les artistes et les créateurs s'inscrivent. Ils peuvent ajouter leurs réseaux sociaux sur leur profil et partager leurs activités avec des images dans différentes catégories.

Sur ce site, seuls les membres peuvent ajouter des images et l'administrateur gère seul les contenus et les membres.

Créativ'Box

I.3. Technologies utilisées

Pour réaliser ce projet, j'ai utilisé les outils, les logiciels et les différents modules suivants :

I.3.1. Pour le Front-end

- Handlebars : un moteur de templating HTML
- TailwindCSS : un framework¹ CSS
- Flowbite : une bibliothèque de composants basée sur le framework TailwindCSS
- JQuery : une librairie JavaScript



I.3.2. Pour le Back-end et la base de données

- Nodejs : un environnement d'exécution JavaScript
- NPM : une librairie de modules (Node Package Manager)
- ExpressJS : un framework pour construire un serveur http NodeJS
- MySQL : une base de données
- PhpMyAdmin : une interface de service de gestion de base de données (SGBD) pour MySQL



¹ Infrastructure d'application

I.3.3. Pour le serveur

- VPS - VDS (OVH) : un serveur privé pour l'hébergement du site sous Linux Ubuntu Server 21.0
- Nginx : un serveur web (Reverse Proxy)
- Docker : une plateforme pour exécuter le projet à l'intérieur d'un conteneur indépendamment.



I.3.4. Les logiciels et les outils

- Git : le système de contrôle de version
- GitHub : un site pour stocker et partager les codes
- Visual studio code : un éditeur de code (IDE) ²



² Environnement de développement

II. LISTE DES COMPÉTENCES DU RÉFÉRENTIEL QUI SONT COUVERTES PAR LE PROJET

II.1. Maquette et diagrammes

Afin de concevoir mon projet, j'ai dû réfléchir à maquetter mon site sous plusieurs aspects, on appelle ça les UML (Unified Modeling Language).

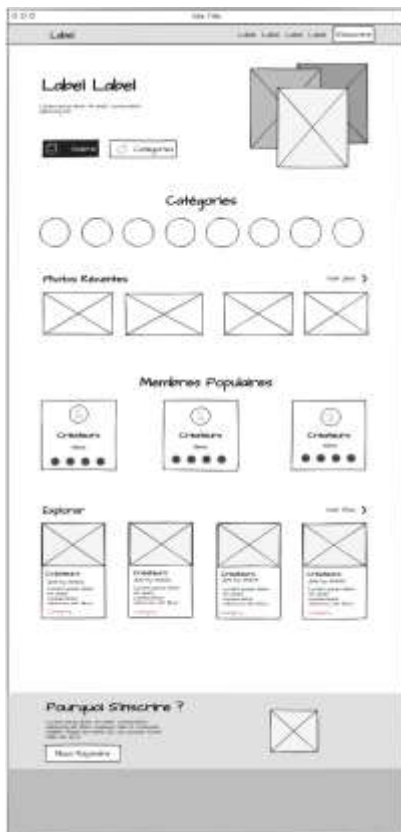


Cela nous permet d'avoir un discours de conception entre les différents collaborateurs qui devront réaliser le projet, il comporte :

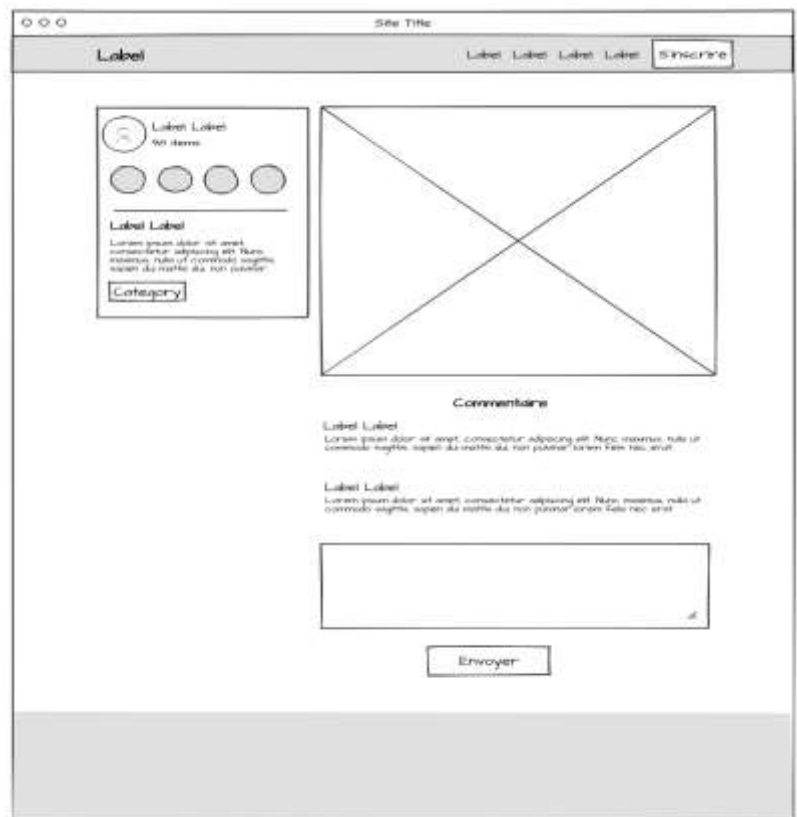
- Une maquette (wireframe) qui me permet de visualiser l'aspect visuel du projet.
- Un diagramme de classe qui m'a permis de modéliser ma base de données.
- Un diagramme de cas d'utilisation (use case) qui m'a permis de visualiser les fonctionnalités du projet suivant les différents rôles utilisateurs disponibles sur le projet.
- Un plan du site³ qui me permet d'avoir une visualisation simple de l'architecture et de l'arborescence de mon site.

³ Sitemap

J'ai utilisé les sites **Mockflow**⁴ pour créer des **maquettes**, **Webflow**⁵ pour créer un plan du site et **Gitmid**⁶ pour créer les diagrammes UML.



Maquette page accueil



Maquette page photo

⁴ www.mockflow.com

⁵ www.gitmind.com/fr

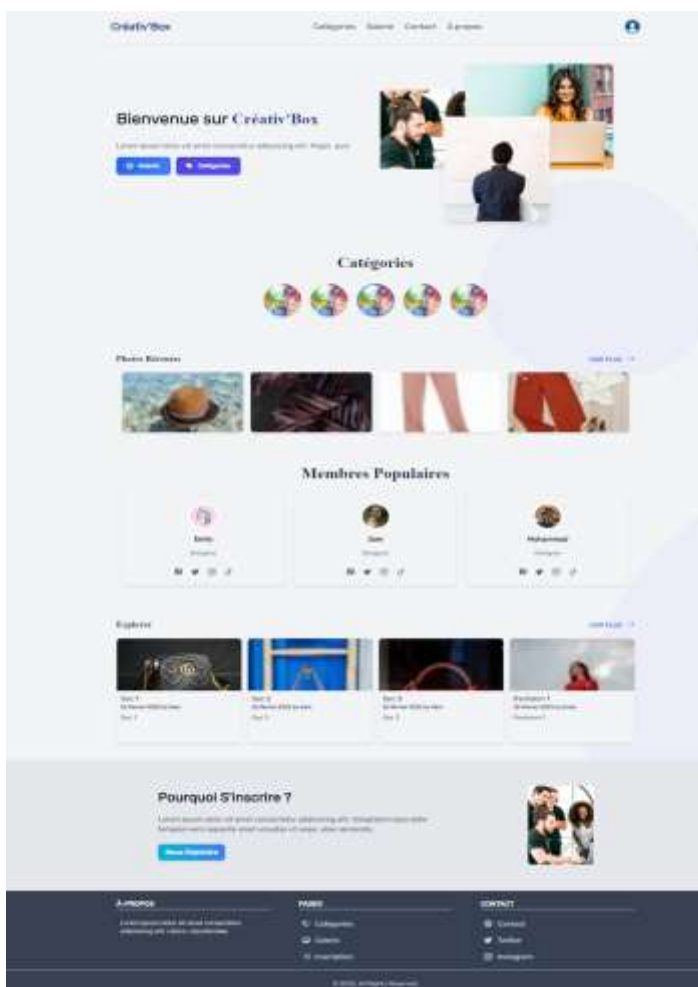
⁶ www.webflow.com

II.2. Réaliser une interface statique et adaptable

À l'aide des wireframes que j'ai créés, j'ai commencé à développer la section Front-end. Aujourd'hui, avec l'expansion de l'utilisation des smartphones, le pourcentage de personnes qui viennent visiter les sites avec leur mobile a augmenté.

Pour cette raison j'ai commencé à créer le site en version mobile (responsive), et à côté de cela, j'ai modifié et optimisé l'apparence du site pour la version bureau.

Pour rendre le site adapté aux téléphones et aux ordinateurs de bureau, j'ai utilisé les propriétés **Flex** et **Grid** en différentes sections. À la place d'écrire les media queries manuellement, j'ai utilisé des classes dans le framework **TailwindCSS**.



Page Accueil (version Bureau)



Page Accueil (version Mobile)

II.3. Développer une interface web dynamique

Pour que les contenus du site puissent être toujours à jour, j'ai récupéré les anciennes données de ma base de données. Pour la page galerie, quand l'internaute effectue une recherche d'image par catégorie ou par membre, le résultat est associé à sa demande.

Pour passer des données aux modaux, j'ai utilisé le framework JQuery. Par exemple, il est utilisé sur les pages profil et administrateur pour modifier ou supprimer des contenus.

J'ai également utilisé **JSDOM** dans mon projet. Par exemple, dans le sidebar de la page de l'administrateur, pour faciliter la navigation dans le menu, il ne peut pas cliquer sur le lien de la page en cours et le lien sera affiché sous une couleur différente.

J'ai utilisé le moteur de templating handlebars (HBS) pour créer un site dynamique. Par exemple pour créer une navbar dynamique j'ai utilisé l'helper « if ... else » pour afficher soit : l'avatar du membre connecté ou alors deux boutons pour s'inscrire ou se connecter.

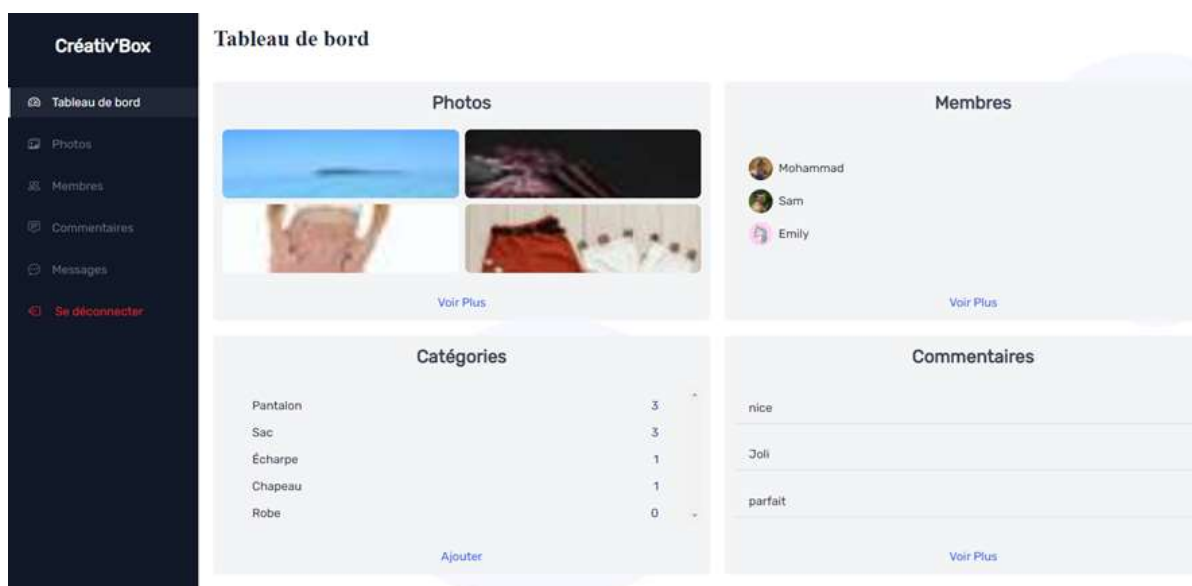
II.4. Réaliser une interface utilisateur et administrateur avec une solution de gestion de contenu et de profil

Les parties les plus importantes du site sont : la possibilité d'envoyer des images et leur gestion par les utilisateurs. À cet effet, une section a été créée sur la page des **membres** pour envoyer des images et une autre section pour gérer celles qui sont publiées. Les membres ont la possibilité de personnaliser leur profil, en modifiant leur avatar, leur nom, les liens de leurs réseaux sociaux.

The image shows a user profile page for 'Créativ'Box'. The page has a light blue header with the site name and navigation links: 'Catégories', 'Galerie', 'Contact', and 'À propos'. A user icon is in the top right. The main content area is titled 'Bienvenue Mohammad' with a placeholder text. It is divided into two columns. The left column, 'Profil', shows a circular profile picture with a 'Select Image' button, followed by input fields for Name (filled with 'Mohammad'), Facebook, Twitter, Instagram, and TikTok, and an 'Edit' button. The right column, 'Charger une photo', has a large image upload area with an upward arrow icon, followed by 'Title' and 'Description' input fields (the description field has the placeholder 'Écrivez votre description'), a 'Robe' dropdown menu, and an 'Envoyer' button. Below these is a 'Liste des photos' section showing two items: 'Chapeau 1' and 'Écharpe 1', each with a small image, a green edit icon, and a red delete icon. The footer is dark blue and contains three sections: 'À-PROPOS' with placeholder text, 'PAGES' with links to 'Catégories', 'Galerie', and 'Inscription', and 'CONTACT' with links to 'Contact', 'Twitter', and 'Instagram'. A copyright notice '© 2022. All Rights Reserved.' is at the bottom center.

Page profil

Une section distincte a été créée pour l'**administrateur** du site pour gérer les contenus du site tel que : les commentaires, les messages, les membres, les images et les catégories.



Page administration

Dans tous les cas, si l'utilisateur est un membre ou bien l'administrateur, s'il souhaite faire des modifications ou des suppressions, on ajoute des modaux et des formulaires pour que ce soit possible.

II.5. Créer une base de données

```

# SCHEMA DUMP FOR TABLE: categorie
#
CREATE TABLE IF NOT EXISTS 'categorie' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'name' varchar(20) NOT NULL,
  'photo' varchar(255) NOT NULL DEFAULT 'creative-pare_17907.jpg',
  PRIMARY KEY ('id'),
  UNIQUE KEY 'id' ('id')
) ENGINE = InnoDB AUTO_INCREMENT = 22 DEFAULT CHARSET = utf8mb4;

#
# SCHEMA DUMP FOR TABLE: comment
#
CREATE TABLE IF NOT EXISTS 'comment' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'text' varchar(255) NOT NULL,
  'photo_id' int(11) NOT NULL,
  'membres_id' int(11) NOT NULL,
  PRIMARY KEY ('id'),
  UNIQUE KEY 'id' ('id'),
  KEY 'photo' ('photo_id'),
  KEY 'user' ('membres_id'),
  CONSTRAINT 'user' FOREIGN KEY ('membres_id') REFERENCES 'membres' ('id')
) ENGINE = InnoDB AUTO_INCREMENT = 10 DEFAULT CHARSET = utf8mb4;

#
# SCHEMA DUMP FOR TABLE: membre
#
CREATE TABLE IF NOT EXISTS 'membres' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'role' tinyint(4) DEFAULT 0,
  'name' varchar(50) CHARACTER SET utf8 DEFAULT NULL,
  'email' varchar(100) CHARACTER SET utf8 NOT NULL,
  'password' varchar(100) CHARACTER SET utf8 NOT NULL,
  'avatar' varchar(255) CHARACTER SET utf8 DEFAULT 'logo-thumbnail.png',
  'facebook' varchar(100) CHARACTER SET utf8 DEFAULT NULL,
  'instagram' varchar(100) CHARACTER SET utf8 DEFAULT NULL,
  'twitter' varchar(100) CHARACTER SET utf8 DEFAULT NULL,
  'tiktok' varchar(100) CHARACTER SET utf8 DEFAULT NULL,
  'token' varchar(255) CHARACTER SET utf8 DEFAULT NULL,
  PRIMARY KEY ('id')
) ENGINE = InnoDB AUTO_INCREMENT = 20 DEFAULT CHARSET = utf8mb4;

#
# SCHEMA DUMP FOR TABLE: message
#
CREATE TABLE IF NOT EXISTS 'message' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'name' varchar(100) NOT NULL,
  'email' varchar(100) NOT NULL,
  'message' varchar(255) NOT NULL,
  'view' tinyint(4) NOT NULL DEFAULT 0,
  PRIMARY KEY ('id')
) ENGINE = InnoDB AUTO_INCREMENT = 11 DEFAULT CHARSET = utf8mb4;

#
# SCHEMA DUMP FOR TABLE: photo
#
CREATE TABLE IF NOT EXISTS 'photo' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'title' varchar(50) NOT NULL,
  'description' varchar(255) DEFAULT NULL,
  'link' varchar(255) NOT NULL,
  'thumbnail' varchar(255) NOT NULL,
  'alt' varchar(100) DEFAULT NULL,
  'membres_id' int(11) NOT NULL,
  'cat_id' int(11) NOT NULL,
  'timestamp' timestamp NOT NULL DEFAULT current_timestamp(),
  PRIMARY KEY ('id'),
  UNIQUE KEY 'id' ('id'),
  KEY 'user' ('membres_id'),
  KEY 'categorie' ('cat_id')
) ENGINE = InnoDB AUTO_INCREMENT = 62 DEFAULT CHARSET = utf8mb4;

#
# SCHEMA DUMP FOR TABLE: sessions
#
CREATE TABLE IF NOT EXISTS 'sessions' (
  'session_id' varchar(128) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL,
  'expires' int(10) unsigned NOT NULL,
  'data' mediumtext CHARACTER SET utf8mb4 COLLATE utf8mb4_bin DEFAULT NULL,
  PRIMARY KEY ('session_id')
) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4;

```

Pour la création d'un site dynamique, il y a besoin d'une base de données pour stocker les contenus du site et utiliser les données où c'est nécessaire. Pour cette raison j'ai utilisé MySQL. Dans ce projet, la base de données à deux fonctions :

- 1- stocker les données
- 2- sécuriser le site.

Afin d'accorder l'accès à certaines pages selon le rôle, on utilise la base de données qui gère les permissions :

- Les visiteurs qui ne se sont pas inscrits sur le site, sont des utilisateurs qui ne sont pas enregistrés sur la base de données. Ils n'ont donc aucuns droits de publications et de modifications. Ils ont uniquement la possibilité de regarder ou de rechercher des publications. Ils peuvent également envoyer des messages à l'administrateur.
- Les utilisateurs qui se sont inscrits ont les mêmes droits, mais peuvent publier leurs images, avoir un profil et écrire des commentaires.

- L'administrateur a tous les droits pour gérer la totalité du contenu du site.

J'ai utilisé un système de gestion de base de données (SGBD) qui comprend **PhpMyAdmin** pour la gestion de l'administration graphique, les **requêtes SQL** (avec PhpMyAdmin et le terminal sur mon VPS) et la **base de données MySQL**.

Pour créer la base de données, j'ai d'abord créé un diagramme de classe pour avoir une compréhension générale des tables, des colonnes et de la relation entre elles. Ensuite, j'ai créé la base de données avec PhpMyAdmin et aussi un compte utilisateur avec les privilèges globaux (**ALL PRIVILEGES**) afin que je puisse accéder à la base de données.

La base de données que j'ai créée est conçue à partir de différentes tables. Il existe des relations entre certaines de ces tables, qui me permet pouvoir appeler des objets d'une table en relation avec une autre.

Pour transformer la base de données que j'ai créée localement sur le serveur VPS, j'ai utilisé un module qui s'appelle **mysqldump** qui permet d'obtenir une copie de sauvegarde (backup) de la base de données. Grâce à ce fichier de sauvegarde, j'ai pu importer ses données sur le serveur VPS.

II.6. Développer les composants d'accès aux données

Sur ma base de données, il y a des relations entre les tables « photo », « membre », « comment » et « category ». Ces relations sont faites par **Foreign Key** dans chaque table. Ainsi, nous pourrions facilement savoir quel membre a posté une image ou laissé un commentaire sur une image en particulier.

J'ai utilisé des **méthodes HTTP** du type **GET**, **POST**, **PUT** et **DELETE** pour gérer la base de données. Grâce à ces méthodes, j'ai pu concevoir un site **CRUD**.

L'acronyme **CRUD** désigne les 4 opérations de base de la gestion des données :

- « Create » : Créer une donnée avec la méthode **POST**.
- « Read » : Lire ou retrouver une donnée avec la méthode **GET**.
- « Update » : Mettre à jour une donnée avec la méthode **PUT**.
- « Delete » : Supprimer une donnée avec la méthode **DELETE**.

La communication entre le Front-end (client) et le Back-end (serveur) est gérée via un système de callback req (request) et res (response) grâce au module ExpressJS. La **requête** est la demande du client au serveur, et la **réponse** est celle du serveur à cette demande.

Les données sont passées via l'**objet req** en trois instances depuis le côté client :

- 1 req.params « `/:id` »
- 2 req.query « `/?key=value` »
- 3 req.body : utilisé particulièrement pour les formulaires de type **POST** ou **PUT** au format **JSON**. La fonction « `express.json()` » est une fonction middleware intégrée dans ExpressJS, précisément avec le module `body-parser`.

Parce qu'il n'est pas possible d'utiliser les méthodes « **DELETE** » et « **PUT** » dans le formulaire, j'ai utilisé le module « **method-override** » qui modifie une méthode POST en une méthode PUT ou DELETE, ce qui permet d'ajouter un query sur les URLs et de transformer notre requête.

```
1 <form class="w-full flex flex-col gap-3"
2     action="/profil/edit?_method=PUT"
3     enctype="multipart/form-data" method="post">
```

Ex : Utilisation de la méthode PUT dans un formulaire.

Cela permet d'utiliser plusieurs méthodes différentes dans une même route.

```
1 Router.route("/profil")
2   .get(isUser, profileGet)
3   .post(isUser, upload.single("photo"), profilePost);
4
5 Router.route("/profil/edit").put(isUser, upload.single("avatar"), profilPut);
6
7 Router.route("/profil/photo").put(isUser, photoEdit);
8 Router.route("/profil/photo/:id").delete(isUser, photoDelete);
9
```


Si nous prenons l'exemple du contrôleur de « modifier le profil », on utilise une fonction asynchrone avec deux callback (req, res). D'abord, il y a une condition si un membre du site soumet cette demande ou non. Ensuite, nous stockons l'ID membre adosser avec le cookie de session, puis on déstructure l'objet `req.body` dans plusieurs constantes. En utilisant ces dernières, nous pouvons effectuer les requêtes SQL qui seront envoyées à la base de données. Cette fonction va retourner un objet. Je lance cette fonction dans une instruction « try...catch » pour contrôler le flux d'instructions et la gestion des erreurs.

```

1 exports.profilPut = async (req, res) => {
2   if (res.locals.user) {
3     const { id } = res.locals.user;
4     const { name, facebook, instagram, twitter, tiktok } = req.body;
5     let db = {};
6     let data = {
7       editUser: null,
8     };
9   }
10
11   db.editProfile = async () => {
12     if (!req.file) {
13       const query = await promiseQuery(
14         'UPDATE membre SET name = `${name}`, facebook = `${facebook}`,
15         instagram = `${instagram}`, twitter = `${twitter}`, tiktok = `${tiktok}` WHERE membre.id = ${id}'
16       );
17       return query;
18     } else {
19       let avatar = `${req.file.destination}/${req.file.filename}`;
20       avatar = avatar.replace('public', 'assets');
21       const query = await promiseQuery(
22         'UPDATE membre SET name = `${name}`, avatar=${avatar}',
23         facebook = `${facebook}`, instagram = `${instagram}`, twitter = `${twitter}`, tiktok = `${tiktok}` WHERE membre.id = ${id}'
24       );
25       return query;
26     }
27   };
28
29   try {
30     await db.editProfile();
31     res.status(200).redirect("/profil");
32   } catch (error) {
33     console.log(error);
34   }
35   else {
36     res.redirect("/connexion");
37   }
38 };

```

II.7. Développer la partie Back-end d'une application web ou web mobile

Pour la partie **Back-end**, j'ai utilisé NodeJS. Certaines fonctions, telles que la connexion à la base de données et la gestion des fichiers, nécessite du temps pour s'exécuter. Pour cette raison, afin de mieux gérer ces fonctions, j'ai utilisé des fonctions asynchrones dans les contrôleurs. J'ai utilisé la propriété **async await** pour

rendre les fonctions asynchrones plus faciles et plus compréhensibles. `async` `await` permet aussi de convertir temporairement les fonctions synchrones en fonctions asynchrones. Si on définit une fonction avec **async**, elle renvoie toujours une promesse et **await** permet d'attendre la résolution d'une promesse et retourne sa valeur.

Sur mon projet, j'ai utilisé les deux modules appelés **multer** et **sharp** pour gérer des images. Multer en tant que middleware, permet aux membres de charger leurs images sur le site et sharp crée une copie plus petite que l'image.

D'autres modules ont été utilisés pour concevoir ce projet, notamment **Nodemailer**, qui est utilisé pour envoyer des e-mails sur le protocole **SMTP**. **bcrypt** est utilisé pour hacher les mots de passe et **rand-token** est utilisé pour créer des jetons lorsqu'un membre demande de réinitialiser son mot de passe.

Enfin, j'ai mis ce projet sur un serveur **VPS** sous le système **Ubuntu**. J'ai utilisé le panel d'administration OVH pour gérer le nom de domaine, puis j'ai utilisé **Nginx** pour le serveur web. J'ai également utilisé **Docker** pour gérer et exécuter mon projet sur VPS.

```

1 ////////////// ? >>> Page Admin >>> Dashboard
2 const { adminGet, addCategory } = require('../controllers/admin/dashboard.controller')
3 Router.route("/admin").get(isAdmin, adminGet);
4 Router.route("/admin/cat").post(isAdmin, addCategory);
5
6 ////////////// ? >>> Page Admin >>> Photos
7 const { adminPhotosGet, adminPhotosPut, adminPhotosDelete } = require('../controllers/admin/photos.controller')
8 Router.route("/admin/photos")
9   .get(isAdmin, adminPhotosGet)
10  .put(isAdmin, adminPhotosPut)
11 Router.route("/admin/photos/:id")
12   .delete(isAdmin, adminPhotosDelete)
13
14 ////////////// ? >>> Page Admin >>> Membres
15 const { adminMembresGet, adminMembresPut, adminMembresDelete } = require('../controllers/admin/membres.controller')
16 Router.route("/admin/membres")
17   .get(isAdmin, adminMembresGet)
18   .put(isAdmin, adminMembresPut)
19 Router.route("/admin/membres/:id")
20   .delete(isAdmin, adminMembresDelete)

```

Exemple d'une partie de mon routeur qui appelle les contrôleurs de la page admin

II.8. Elaborer et mettre en œuvre des composants de gestion de contenu

La gestion des contenus est gérée avec CRUD (Create, Read, Update, Delete).

J'ai créé une page réservée pour les membres afin qu'ils puissent gérer leurs contenus. Ils peuvent charger, modifier et supprimer leurs propres images. Ils peuvent également mettre à jour leur profil.



Gestion des images publiées sur le profil membre

Dans le cas de la gestion des images publiées sur la page profil d'un membre (exemple ci-dessus), je récupère l'id de ce dernier afin que les modifications ou les suppressions apportées soient uniquement en lien avec ce membre.

```

1  [[@each userphotos]]
2  <div class="flex justify-between gap-5 border-b border-gray-100 py-3">
3    <div class="flex gap-4 items-center flex-1">
4      
5      <a class="text-blue-600 font-sans font-weight-normal text-sm mt-1" href="/galerie/{{id}}">
6        {{title}}
7      </a>
8    </div>
9    <div class="flex flex-direction column space-x-2">
10     <button data-id="{{id}}" data-title="{{title}}" data-description="{{description}}" data-category="{{cat_id}}"
11       data-action="/profil/photo?_method=PUT" data-modal-toggle="edit-modal"
12       class="edit-modal bg-green-200 hover:bg-green-100 rounded-lg animation p-6">
13       <i class="bi bi-pencil-fill text-green-900"></i>
14     </button>
15     <button data-thumbnail="{{thumbnail}}" data-action="/profil/photo/{{id}}?_method=DELETE"
16       data-modal-toggle="delete-modal" class="bg-red-200 p-4 hover:bg-red-100 rounded-lg animation delete-modal">
17       <i class="bi bi-x-lg w-20 text-red-900"></i>
18     </button>
19   </div>
20 </div>
21 [[/each]]

























```

Boucle « each » qui récupère les images publiées par un membre

Pour l'administrateur, j'ai créé une page que lui seul peut avoir accès.
Il a tous les droits pour créer, modifier, supprimer des contenus (images, commentaires, messages, catégories et membres).

#	Photo	Nom	Modifier	Supprimer
11		Emel		
18		Sam		
19		Mohammad		

Gestion des membres sur la page admin

#	Photo	Titre	Membres	Catégories	Modifier	Supprimer
44		Sac 1	Sam	Sac		
45		Sac 2	Sam	Sac		
46				Sac		
47				Portakal		
48				Portakal		
49				Portakal		
50		Écharpe 1	Mohammad	Écharpe		
51		Chapeau 1	Mohammad	Chapeau		

Modal pour supprimer une image

```

1 exports.adminPhotosDelete = async (req, res) => {
2   const { deletePhoto } = require('../helpers/deletePhoto')
3   const { id } = req.params
4   const { user, role } = res.locals.user
5
6   try {
7
8     let remove = await deletePhoto(req, res, id, user, role, '/admin/photos')
9
10    if (remove) {
11      console.log('Photo Was deleted')
12    }
13  } catch (error) {
14    console.log(error)
15  }
16 }

```

Contrôleur pour supprimer une image

```

1  const promiseQuery = require('.../Database/Database');
2  const fs = require('fs')
3
4  exports.deletePhoto = async (req, res, id, user, role, redirect) => {
5
6    let profil, admin = {}
7
8    async function removePhoto(photo, thumbnail) {
9
10     photo = await photo.replace('assets', 'public')
11     thumbnail = await thumbnail.replace('assets', 'public')
12
13     await fs.unlink(photo, async (err) => {
14       if (err) throw err
15     })
16
17     await fs.unlink(thumbnail, async (err) => {
18       if (err) throw err
19     })
20   }
21
22   if (role === 'U') {
23     profil = await promiseQuery('select * from photo where photo.id = ?[id] and photo.user_id = ?[user]', async (err, results) => {
24       if (err) throw err
25       await removePhoto(results[0].link, results[0].thumbnail)
26       await promiseQuery('DELETE FROM photo WHERE photo.id = ?[id]', (err) => {
27         if (err) throw err
28         res.status(200).redirect(`${redirect}`)
29       })
30     })
31     return profil
32   } else if (role === 'A') {
33     admin = await promiseQuery('select * from photo where photo.id = ?[id]', async (err, result2) => {
34       if (err) throw err
35       await removePhoto(result2[0].link, result2[0].thumbnail)
36       await promiseQuery('DELETE FROM photo WHERE photo.id = ?[id]', (err) => {
37         if (err) throw err
38         res.status(200).redirect(`${redirect}`)
39       })
40     })
41     return admin
42   }
43 }

```

Module pour supprimer une image

III. CAHIER DES CHARGES

III.1. Fonctionnement de l'application

III.1.1. Permissions

Dans ce projet, il existe trois types d'utilisateurs avec des droits différents :

1. **Les visiteurs** : ils ont uniquement la possibilité de regarder ou de rechercher des publications, envoyer des messages à l'administrateur et s'inscrire sur le site.
2. **Les membres** : ils ont tous les droits des visiteurs. Ils peuvent également publier leurs images, mettre des commentaires et avoir un profil.
3. **L'administrateur** : Il a tous les droits pour gérer la totalité du contenu du site.

III.1.2. Fonctionnalités en fonction des permissions

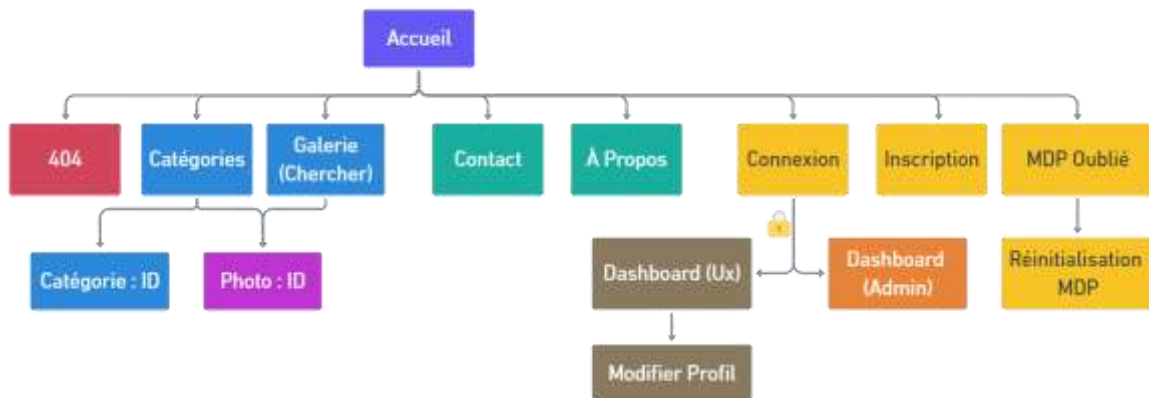
- **Les visiteurs** :
Accueil - à-propos - Cat (catégories) - Cat/:id - Galerie - Galerie/:id (photo) -
Contact - Inscription - Connexion
- **Les membres** :
Accueil - à-propos - Cat (catégories) - Cat/:id - Galerie - Galerie/:id (photo) -
Contact - Inscription - Connexion
+ Profil
- **L'administrateur** :
Accueil - à-propos - Cat (catégories) - Cat/:id - Galerie - Galerie/:id (photo) -
Contact - Inscription - Connexion
+ admin/tableau-de-bord - admin/photos - admin/messages - admin/commentaires
- admin/membres - admin/messages

III.1.3. Courte description de chaque URL

- **/ (accueil)** : Présentation de l'objectif du site. Les visiteurs ont un accès rapide au contenu grâce aux raccourcis (images récentes, membres populaires, ...).
- **/cat** : Accès principal aux catégories existantes. Elles ont chacune un affichage des 4 dernières images publiées et un lien pour la totalité de leur contenu.
- **/cat /name** : Afficher toutes les images de chaque catégorie.
- **/galerie** : Affichage de l'ensemble des images publiées avec la possibilité d'effectuer une recherche par critère.
- **/galerie/:id** : Afficher l'image avec ses détails comme le titre, la description, la catégorie et son auteur. Présence d'un formulaire pour envoyer des commentaires.
- **/à-propos** : Description du site et explications de ses objectifs.
- **/contact** : Formulaire pour envoyer un message à l'administrateur du site.
- **/connexion** : Formulaire d'identification du membre pour se connecter sur son profil.
- **/inscription** : Formulaire d'inscription pour devenir un membre.
- **/mot-de-passe-oublié** : Formulaire de demande de réinitialisation du mot de passe.
- **/profil** : Page de profil des membres, avec la gestion pour modifier le profil, envoyer des images et modifier ou supprimer les images publiées.
- **/admin** : Espace de l'administrateur du site. Avec la gestion des images, membres, messages, commentaires et ajout des catégories.

III.2. Arborescence du site

Cette arborescence a été créée au début du projet. Des changements ont été apportés dans les versions ultérieures.



IV. SPÉCIFICATIONS FONCTIONNELLES

IV.1. Introduction

Une première liste des fonctions a été écrite sur le cahier des charges. Cependant, il est nécessaire d'établir une liste détaillée (voir le tableau des fonctions ci-dessous). Dans le cas de prochaines évolutions (ajout, modification ou suppression de pages), de nouvelles versions de tableau seront présentées.

IV.2. Tableau des fonctions

Page	Route	Méthode	Permission	Observation
Accueil	/	Get	Tous	Page d'accueil avec la récupération de données de la BD
Catégories	/cat	Get	Tous	Page des catégories avec la récupération des catégories avec 4 images récentes de la BD
Catégories	/Cat /:name	Get	Tous	Page de la catégorie avec la récupération des images de chaque catégorie de la BD
Galerie	/galerie	Get	Tous	Page de la galerie avec la récupération des images de la BD
Galerie	/galerie	Post	Tous	Effectuer une recherche par critère
Photo	/Galerie/:id	Get	Tous	Page photo avec la récupération des détails et des commentaires de la BD

Photo	/Galerie/:id	Post	Membre / admin	Possibilité de laisser un commentaire
Contact	/contact	Post	Tous	Envoyer un message à l'administrateur
Connexion	/connexion	Get	Visiteurs	Accès à la page de connexion
Connexion	/connexion	Post	Visiteurs	Se connecter avec la récupération des infos dans la BD / Création de sa session et de son cookie
Inscription	/inscrire	Post	Visiteurs	Création d'un compte
MDP Oublié	/mot-de-passe-oublie	Post	Visiteurs	Faire une demande de réinitialisation de mot de passe et envoyer le token
Réinitialisation mot de passe	/reinitialisation-mot-de- passe	Post	Visiteurs	Réinitialisation du mot de passe
Déconnecter	/deconnecter	Get	Membre / Admin	Déconnecter et supprimer la session et le cookie
Profil	/profil	Get	Membre	Compte utilisateur avec la récupération d'infos dans la DB
Profil	/profil	Post	Membre	Publier une image, son titre, sa description et sa catégorie
Profil	/profil/edit	Put	Membre	Édition d'un membre, avatar, nom et les liens des réseaux sociaux
Profil	/profil/photo	Put	Membre	Édition d'une image, son titre, sa description ou sa catégorie

Profil	/profil/photo/:id	Delete	Membre	Supprimer une image sur la base de données et l'hébergement
Admin	/admin	Get	Admin	Permet à l'admin d'avoir accès au panneau admin
Admin	/admin	Post	Admin	Ajouter d'une catégorie
Admin	/admin/photo	Put	Admin	Modifier les détails d'une image
Admin	/admin/membres/:id	Delete	Admin	Suppression d'un membre
Admin	/admin/commentaires/:id	Delete	Admin	Suppression d'un article via son id

V.SPÉCIFICATIONS TECHNIQUES

V.1.Stockage et version de contrôle

J'ai utilisé **Git** et **GitHub** pour gérer ce projet.

Git est un système de contrôle de versions qui permet d'avoir un historique des modifications de code ou de séparer le code en d'autres branches.



GitHub est un site pour stocker ou partager nos codes. En utilisant les commandes de Git tel que : "**add**", "**commit**" et "**push**", j'ai transféré le projet sur un **repository** sur GitHub. En utilisant "**clone**" et "**pull**", j'ai téléchargé la dernière version de mon projet sur mon ordinateur ou VPS.



En raison de la possibilité d'avoir des bogues lors du développement du projet, j'ai créé plusieurs branches. Au début, je développais le projet sur la branche **DEV** et quand je corrigeais les bogues, je fusionnais cette branche avec la branche **MAIN** (checkout, status, branch, ...).

V.2.Workflow

Pendant ce projet, j'ai utilisé les conseils de mon formateur ou de mes camarades. S'il y avait un problème sur mon code, j'utilisais leurs conseils ou bien j'effectuais une recherche sur Internet pour trouver la solution.

Pour mettre mon site en production, j'ai acquis un serveur VPS. J'ai également utilisé **NPM** pour utiliser les packages **Node** pour exécuter mon site sur le VPS.



V.3.Les technologies utilisées

V.3.1.Front-end

Pour la section **Front-End**, j'ai utilisé les technologies apprises lors de cette formation, ainsi que d'autres technologies mises à jour tel que **TailwindCSS**, **Flowbite**, **colcade**, **scrollReveal**, **animate** pour améliorer la qualité **UX/UI** du site. J'ai également utilisé **Handlebars**, qui est un moteur de templating. Il m'a aidé à écrire moins de code mais plus efficacement. J'ai utilisé ses **helpers** comme « **if** » et « **each** » pour avoir un site dynamique.



V.3.2.Back-end

Pour la section **Back-End**, j'ai utilisé les technologies apprises lors de cette formation comme **NodeJS**, qui est un environnement de JavaScript.

Pour la création du serveur avec NodeJS, j'ai utilisé le framework **ExpressJS**.

J'ai utilisé d'autres modules : je peux appeler "**Method-override**" qui me permet d'utiliser les méthodes "**PUT**" et "**DELETE**" sur les formulaires. "**multer**" et "**charp**" pour la gestion des images. J'ai également utilisé "**Nodemailer**" pour envoyer des e-mails, par exemple pour changer le mot de passe ou permettre à l'administrateur de répondre aux messages. J'ai également utilisé « **rand-token** » pour créer des tokens pour la réinitialisation des mots de passe, **bcrypt** pour hacher les mots de passe et **dotenv** pour cacher les valeurs de certaines variables spécifiques (variables d'environnement).



express



V.3.3. Base de données

Comme nous l'avons déjà vu dans ce projet, j'ai utilisé la base de données **MySQL** et le système de gestion de base de données MySQL, composé de **PHPMyAdmin** pour l'administration graphique et la modélisation de la base de données. J'ai également utilisé le terminal MySQL, le langage SQL et le module NPM « **mysql** » comme passerelle.

Dans ce projet j'ai utilisé les modules **express-session** et **express-mysql-session** qui me permettent de créer une table sur la base de données avec le nom « **session** » qui contient les données de connexion de l'utilisateur.

J'ai par exemple utilisé un module NPM appelé "**mysqldump**" afin de pouvoir effectuer une sauvegarde de la base de données.



V.3.4. Tests unitaires

Pour réaliser les tests unitaires, j'ai utilisé le framework **Mocha**, ainsi que la bibliothèque **CHAI** et les modules **chai-http** et **assert()**.

Avec **chai** et **chai-http**, on peut faire des tests unitaires afin de tester les routeurs. En utilisant **assert()**, on peut comparer les résultats.



V.3.5. Gestion de documentations API

Avec le module **express-oas-generator**, j'ai récupéré toutes les routes **api** de mon site, avec leurs méthodes (HTTP). Les résultats sont accessibles sur les routes **/api-docs/v2** ou **/api-docs/v3**.

J'ai enregistré ces résultats dans un fichier **.json**. Grâce à ce fichier et le module **swagger-ui-express**, j'ai créé la documentation API sur la route **/api-docs**.



VI. RÉALISATION DU PROJET ET DU DÉVELOPPEMENT

VI.1. Introduction

Dans cette section, il traite la période de 6 mois dans la formation de développeur web et web mobile et du développement de ce projet.

VI.2. L'environnement

Ma formation est située à Allonnes, au sein de la Maison des Arts. L'environnement de la formation était intime et charmant et les camarades de classe avaient une bonne relation amicale.

Parallèlement à notre formation, la formation CDA se tenait en même temps et les étudiants nous ont honnêtement parlé de leurs expériences.

Les ordinateurs de la formation étaient de bonne qualité et je n'ai eu aucun problème avec eux, pendant la période de formation.

Je tiens à remercier **ARINFO** de m'avoir donné l'opportunité de participer à ces cours. Je remercie tout particulièrement mon formateur **Morgan BRISSON** qui compte tenu de ma situation, a accepté d'enseigner les cours de formation et nous a fait profiter de ses connaissances et de son expérience.

Je tiens également à remercier ma conjointe de m'avoir aidé tout au long de cette période et sans son aide, j'aurais certainement eu beaucoup de problèmes. Je tiens également à remercier **Etic Asso** de m'avoir aidé à participer à ces cours.

VI.3. Pourquoi une formation dans le développement ?

Je m'intéresse aux jeux d'esprit depuis mon enfance et j'aime résoudre des problèmes de logique. Cet intérêt m'a amené à passer beaucoup de temps avec un ordinateur et Internet. Grâce à cela, je me suis familiarisé avec la programmation.

En raison des restrictions qui existent en Iran pour les étrangers, je n'ai pas pu poursuivre mes études dans ce domaine. Après avoir immigré en France et être autorisé à y rester, j'ai aujourd'hui la chance de poursuivre mon objectif en participant à ces cours.

VI.4. Organisation

L'organisation est l'un des principes les plus importants dans ce métier. Le cycle en V et les méthodes AGILE ont été inventés à cet effet. Conscient de l'importance de cet enjeu, j'ai essayé de mener à bien ce projet de manière organisée. Cependant, malgré le fait d'être débutant et d'être seul, j'ai dû faire face à des défis. Grâce à beaucoup d'efforts et aux conseils de mon formateur, j'ai pu gérer ses exigences et mener à bien le projet aussi loin que je le souhaitais. J'utiliserai les compétences que j'ai acquises sur l'organisation au cours de ce projet pour mieux travailler et de manière plus organisée.

VI.5. Rythme

Le temps de cette formation a été rapide et intense et nous avons dû acquérir de nombreuses compétences au cours de cette période. Chaque étudiant a besoin de passer plus de temps au-delà de la formation pour améliorer ses compétences. Pour cela, j'ai passé mon temps libre, la nuit et le week-end à m'entraîner et à améliorer mes compétences. Par exemple, une compétence que j'ai apprise en dehors de ce cours est le framework TailwindCSS que j'ai utilisé dans ce projet.

VI.6. La phase de développement

Pour réussir à finaliser un projet proprement et complètement, il est indispensable d'être bien organisé, d'être méthodique et d'avoir la capacité d'anticiper. Ce sont les qualités indispensables d'un développeur.

VI.6.1. Planification

Pour travailler le plus efficacement possible et respecter les délais pour aboutir au projet, j'ai écouté les conseils de mon formateur et j'ai établi une ligne de conduite en prenant en compte les étapes clés de la création d'un site web (Sitemap, Wireframe, diagrammes UML, développement Front-end, création BDD, développement Back-end, ...). Pour contrôler le processus de ce projet, j'ai utilisé Trello (*réf. VI.6.5 les outils*).

VI.6.2. Lancement

Avec les conseils de mon formateur, j'ai débuté mon projet avec l'objectif de terminer une première version stable qui sera présentée au jury de l'examen.

Des fonctionnalités supplémentaires pour enrichir le site web seront intégrées dans de nouvelles versions.

VI.6.3. Déroulement

Pendant la durée de mon projet, j'ai pu rencontrer certaines difficultés. Pour pouvoir les résoudre, j'ai su m'adapter pour les contourner. En souhaitant être un développeur perfectionniste et travailler avec une qualité très poussée, j'ai perdu du temps pour enrichir au maximum mon code. J'ai très rapidement compris qu'il n'était pas nécessaire d'aller aussi loin dans la perfection et je me suis donc concentré uniquement sur l'avancement et l'aboutissement de mon projet, sous la forme d'une première version.

VI.6.4. Finalisation

Je suis très satisfait d'avoir terminé la première version de mon site web. Je suis également très content d'avoir enrichi mes compétences mais également mon niveau de français, malgré les difficultés que j'ai pu rencontrer, grâce au soutien et à la compréhension de mon formateur, de mes camarades et de ma conjointe.

VI.6.5. Les outils

GitHub

C'est un site important et largement utilisé par tous les développeurs. Ils utilisent ce site pour gérer leurs projets ou les partager. Les entreprises utilisent ce site pour faciliter leur travail en équipe. Je connais ce site depuis quelques années, mais je l'ai utilisé quotidiennement pendant la période de cette formation.



Trello

J'ai pris connaissance de ce site au début de la formation. Je l'ai d'abord utilisé pour organiser les cours de formation. Quand j'ai commencé à développer mon projet, je l'ai utilisé pour organiser mes idées ou les activités que je devais faire pour créer mon site. J'ai créé plusieurs colonnes nommées : « à faire », « en cours », « terminés ». J'ai d'abord ajouté les idées dans la colonne « à faire ». Lorsque j'ai travaillé sur chaque idée, je les déplaçais dans la colonne « en cours ». Une fois qu'elles étaient finies, je les déplaçais dans la colonne « terminés ». Cela m'a permis de voir l'avancement du projet.



Discord

Un outil utile pour communiquer avec d'autres personnes. J'ai commencé à l'utiliser au début de la formation. La possibilité de catégoriser les sujets et de créer différentes salles, m'a aidé à bien organiser les sujets et pouvoir accéder plus rapidement à un sujet spécifique.



VII. DÉROULEMENT DU PROJET

VII.1. Introduction

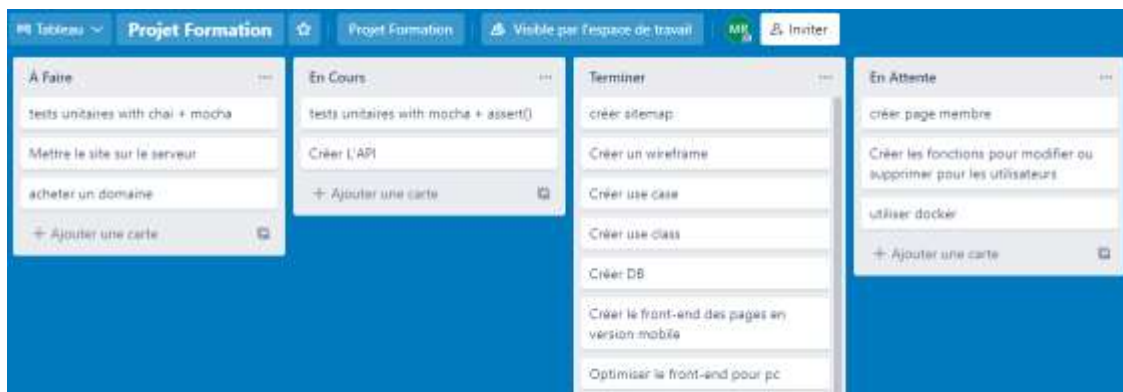
L'idée initiale du projet est de permettre aux créateurs de partager leurs activités et aux visiteurs d'apprendre à les connaître, voir leurs travaux et pouvoir les trouver et les suivre sur d'autres réseaux sociaux tels que Facebook et Instagram.

Ayant eu peu de temps à réaliser un projet qui a été commencé à partir de zéro, j'ai essayé de réaliser la totalité de la création de ce site. J'avais quelque chose de réussi à présenter.

Bien sûr, avec la suggestion de mon formateur, j'ai mis de côté certaines fonctionnalités du site et elles seront ajoutées dans les prochaines mises à jour.

VII.2. Déroulement du projet semaine par semaine

Cette section donne un aperçu du développement du projet, étape par étape :



- **Semaine 1**

Réalisations :

- Sitemap
- Wireframes (maquette)
- Diagrammes UML (use case - use class)
- Création des pages en HTML

Grâce aux maquettes et au framework TailwindCSS, la création de la section Front-End du site a été réalisée en toute simplicité.

- **Semaine 2**

Réalisations :

- Création de la base de données
- Création du routeur et des contrôleurs
- Création de Front-End avec handlebars

À l'aide du diagramme de classes, j'ai créé la base de données avec PhpMyAdmin. Avec le module MySQL, j'ai réalisé la connexion entre la partie Back-End du site avec la base de données. Ensuite j'ai créé le routeur et les contrôleurs et j'ai connecté les contrôleurs à la section de vue.

- **Semaine 3**

Réalisations :

- Utilisation express-session et express-mysql-session
- Création d'un middleware Authentification
- Utilisation bcrypt
- Utilisation Nodemailer pour la page reset password
- Ajouter un middleware auth pour les pages profil et admin
- Création du CRUD dans la page Profil
- Utilisation de multer et sharp pour envoyer des images.

Avec les modules express-session et express-mysql-session, j'ai enregistré les données de connexion des membres. Avec ces données, j'ai créé un middleware qui contrôle l'accès des pages pour chaque rôle. J'ai ensuite utilisé le module bcrypt pour hacher les mots de passe et Nodemailer pour envoyer un mail de demande de réinitialisation de mot de passe ou permettre à l'administrateur de répondre aux messages. J'ai créé un CRUD dans la page profil pour que chaque membre puisse récupérer, envoyer, modifier ou supprimer leurs images ou leur profil. Enfin, le module multer permet aux utilisateurs d'envoyer leurs images.

- **Semaine 4**

Réalisations :

- Création de la page admin
- Création du CRUD dans la page admin
- Création de la page logout
- Création de la page reset password
- Création de la page 404
- Utilisation de JQuery pour passer les données sur les modaux

J'ai tout d'abord voulu créer des onglets dans la page admin, mais il y avait des problèmes pour récupérer les données, alors j'ai été obligé de recréer le Front-End de la page admin. J'ai donc créé un Sidebar pour que l'administrateur puisse accéder à chaque contenu dans des pages différentes. J'ai créé un CRUD pour chacune de ces pages.

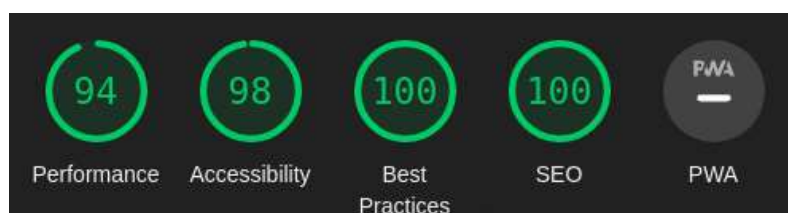
J'ai créé une page de déconnexion pour que chaque membre ou administrateur puisse quitter son compte et enfin, j'ai créé une page 404.

- **Semaine 5**

Réalisations :

- Tests unitaires
- Swagger (API)
- Lighthouse
- Améliorer UI / UX

J'ai réalisé des tests unitaires avec les modules Mocha et chai. J'ai également utilisé le module swagger-ui-express pour créer ma documentation API. J'ai utilisé Lighthouse qui m'a aidé à améliorer la qualité des pages web.



- **Semaine 6**

Réalisations :

- Optimiser les codes
- Tester les codes
- Mettre le site sur VPS

Sur la base des compétences que j'ai acquises lors de la création de ce projet, j'ai essayé d'améliorer mon code et d'utiliser l'architecture **MVC** pour séparer et éviter les doublons.

VIII. PRÉSENTATION DE LA FONCTIONNALITÉ LA PLUS REPRÉSENTATIVE

VIII.1. Introduction

L'une des parties les plus importantes de ce site est de donner la possibilité aux membres de partager des images. Étant donné que j'ai eu peu d'occasions de créer un site avec toutes les idées que j'avais, je me suis concentré sur les sections principales du site et d'ajouter mes autres idées pour les prochaines versions.

La partie que je veux présenter est le **SCRUD** pour les images.

Les utilisateurs doivent pouvoir les envoyer mais doivent aussi comme l'administrateur, les modifier et les supprimer.

Les visiteurs quant à eux, doivent avoir la possibilité d'accéder aux images publiées ou de pouvoir les rechercher.

Chaque image doit avoir une page dédiée avec un format plus petit avec d'autres détails tels que le nom de l'auteur, le titre, la description et la catégorie spécifique.

Les commentaires de chaque image, peut-être affichés s'ils existent. Pour que ce soit possible, il faut établir une connexion entre les différentes tables de la base de données.

VIII.2. Interface utilisateur et administrateur

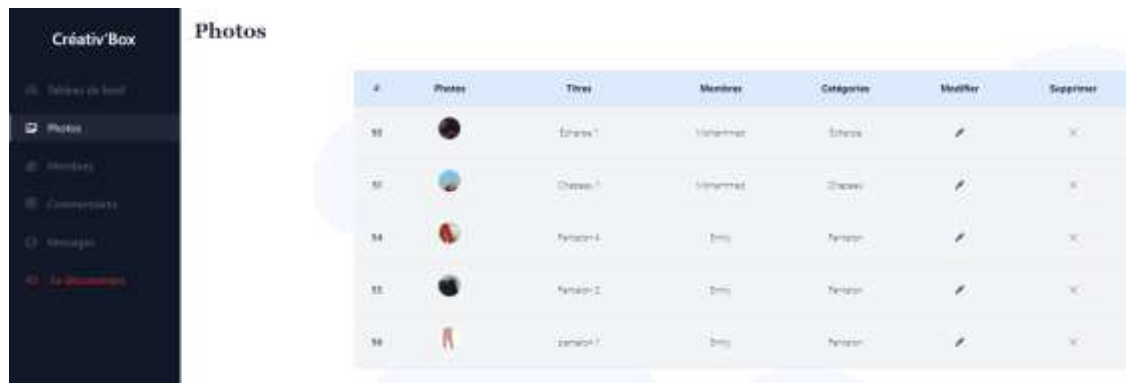
Voici la présentation de la page profil qui permet de réaliser le CRUD au niveau des images, pour les membres :

The screenshot shows a user profile interface. On the left, the 'Profil' section displays a user's name 'Mohammad' and social media links for Facebook, Twitter, Instagram, and TikTok, each with an input field. A 'Select Image' button is below the profile picture. On the right, the 'Charger une photo' section contains a form with a title input, a description input (placeholder: 'Écrivez votre description'), a 'Robe' dropdown menu, and an 'Envoyer' button. A large orange box with the number '1' is overlaid on the top right of this section. At the bottom, the 'Liste des photos' section shows two items: 'Chapeau 1' and 'Echarpe 1', each with a green edit button and a red delete button. A large orange box with the number '2' is overlaid on the middle of this section.

1 : formulaire pour publier une image

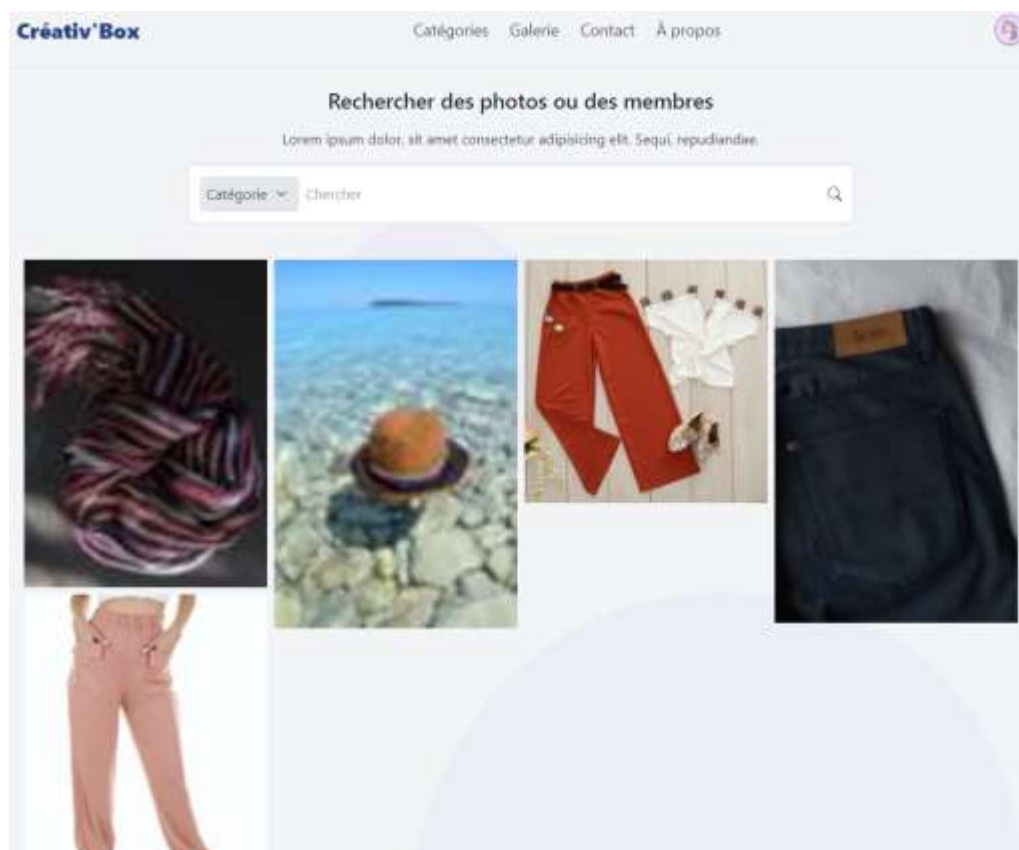
2 : section avec deux boutons d'édition (modification et suppression)

Voici la présentation de la page administrateur qui permet de réaliser le RUD au niveau des images pour l'administrateur :



Cette page affiche toutes les images envoyées sur le site.

Voici la présentation de la page galerie où il existe une fonction de recherche, où les visiteurs peuvent rechercher des images par catégorie ou par membre :



VIII.3. Comment l'utiliser

Seuls les membres ont la possibilité d'envoyer des images. Il y a donc un CRUD sur la page de profil, pour que les membres puissent envoyer leurs images. Lors de la création de leurs publications, ils peuvent ajouter un titre, une description et choisir une catégorie.

Il y a aussi une autre section sur cette page où les membres peuvent modifier ou supprimer leurs images.

Dans la première version du site, il n'y a pas la possibilité pour l'administrateur d'envoyer des images, il peut cependant accéder à la liste de toutes les images publiées dans un tableau sur la page d'administration. Il peut modifier les détails (titre, description, catégorie) ou supprimer des images.

Ensuite, pour trouver des images, les utilisateurs peuvent accéder à la page de la galerie et trouver toutes les images par ordre de publications (les dernières images sont en début de liste). Cette page a la possibilité de rechercher des images par catégorie et par le membre qui les a publiées.

Il existe un lien entre les tables des images, des membres, des commentaires et des catégories. Cette connexion est établie grâce aux **clés externes** « **membre_id** » et « **cat_id** » dans la table 'photo' et « **membre_id** » dans la table 'comment'.

Par exemple, cette relation est utilisée dans la page photo :

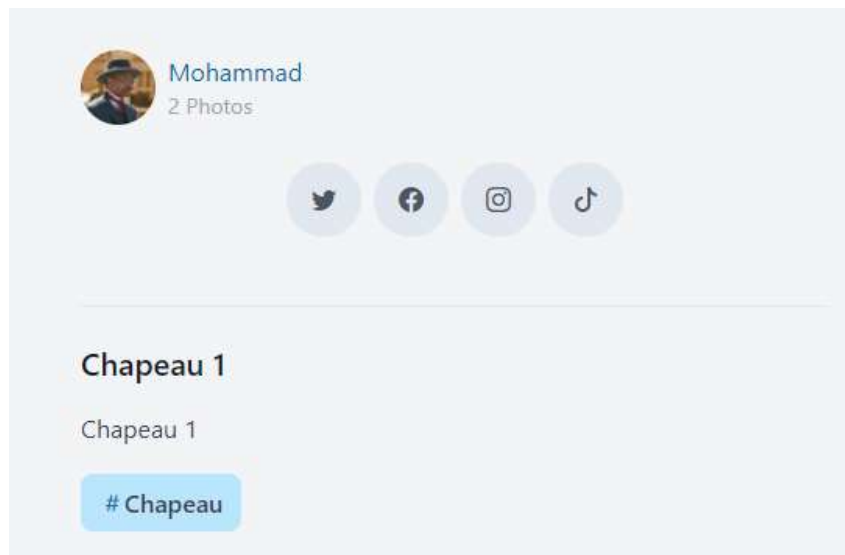
```

1 photoSelectDetails: async (id) => {
2   return await query(`SELECT photo.title,
3     photo.description , membre.name AS membre,
4     membre.avatar, membre.facebook,
5     membre.instagram, membre.twitter,
6     membre.tiktok, categorie.name AS category
7   FROM photo RIGHT OUTER JOIN membre
8     categorie.id = photo.cat_id
9   WHERE photo.id = ${id}`)
10 },

```

Cette page affiche le nom de l'auteur et ses liens vers les réseaux sociaux dans le menu de description.

En bas se trouvent le titre de l'image et sa description. La catégorie est affichée sous la forme d'un bouton, sur laquelle on peut cliquer pour accéder à la page d'une catégorie spécifique.



Il y a une section en bas de page pour poster un commentaire et on peut accéder au nom de l'auteur d'un commentaire et au contenu de son message.

Commentaires

admin
nice

Écrire un commentaire

Écrivez votre commentaire

Envoyer

VIII.4. Comment cela fonctionne

Pour créer un CRUD, j'ai d'abord ajouté les routes, via mon `router.js` :

```
const { profileGet, profilePost, profilPut, photoEdit, photoDelete } = require('../controllers/profile.controller')
Router.route("/profil*")
  .get(isUser, profileGet)
  .post(isUser, upload.single("photo"), profilePost);

Router.route("/profil/edit").put(isUser, upload.single("avatar"), profilPut);

Router.route("/profil/photo*").put(isUser, photoEdit);
Router.route("/profil/photo/:id").delete(isUser, photoDelete);
```

router page profil

```
##### ? >>> Page Admin >>> Dashboard
const { adminGet, addCategory } = require('../controllers/admin/dashboard.controller')
Router.route("/admin").get(isAdmin, adminGet);
Router.route("/admin/cat").post(isAdmin, addCategory);

##### ? >>> Page Admin >>> Photos
const { adminPhotosGet, adminPhotosPut, adminPhotosDelete } = require('../controllers/admin/photos.controller')
Router.route("/admin/photos")
  .get(isAdmin, adminPhotosGet)
  .put(isAdmin, adminPhotosPut)
Router.route("/admin/photos/:id")
  .delete(isAdmin, adminPhotosDelete)
```

router page admin/ et admin/photos

Pour publier les images, j'ai créé un formulaire sur la page de profil afin que les membres puissent publier leurs images.

J'ai créé deux modules pour éditer et supprimer des images, puis j'ai importé ces modules dans les contrôleurs de page d'administration et de profil pour éviter les doublons d'écriture de code.

J'ai utilisé **jQuery** pour le modal « modifier l'image » et avec son aide, j'ai transféré les données de chaque image sur ce modal pour que le membre ou l'administrateur puissent voir ou modifier ces informations.

Pour supprimer l'image correspondant à la méthode **DELETE**, un simple modal suffit pour faire une demande de confirmation de suppression.

Une fois les modaux réalisés, il a fallu créer les actions de la route via mon contrôleur admin ou profil.

```

1  exports.adminPhotosGet = async (req, res) => {
2    const {photoSelectWithCatMembre} = require('../../database/models/model.photos')
3    const {categorySelectAll} = require('../../database/models/model.categories')
4    let data = {}
5    try {
6      data.photos = await photoSelectWithCatMembre()
7      data.categorie = await categorySelectAll()
8
9      data.photos = JSON.parse(JSON.stringify(data.photos))
10     data.categorie = JSON.parse(JSON.stringify(data.categorie))
11     res.status(200).render("pages/admin/Photos", {
12       layout: "blank",
13       photos: data.photos,
14       listCategories: data.categorie,
15       title: 'CreativBox | Photos'
16     });
17   } catch (error) {
18     console.log(error)
19   }
20 };
21
22 exports.adminPhotosPut = async (req, res) => {
23   const {photoPut} = require('../../database/models/model.photos')
24   const { role } = res.locals.user
25   const user_id = res.locals.user.id
26   const data = req.body
27   try {
28     await photoPut(req, res, user_id, role, data)
29     console.log('photo was edited')
30   } catch (error) {
31     console.log('Error', error)
32   }
33 }
34
35 exports.adminPhotosDelete = async (req, res) => {
36   const { photoDelete } = require('../../database/models/model.photos')
37   const { id } = req.params
38   const { user, role } = res.locals.user
39
40   try {
41     await photoDelete(req, res, id, user, role, "/admin/photos")
42   } catch (error) {
43     console.log(error)
44   }
45 }

```

Contrôleur photo page administrateur

Nous devons également avoir un formulaire pour envoyer les demandes au serveur. J'ai donc utilisé le module **method-override** pour pouvoir utiliser les méthodes PUT et DELETE dans les formulaires.

```

1 <form id="getActionEdit" action="" method="post" class="modal-body">
2   <div class="p-6 space-y-6">
3     <label class="block w-full">
4       <span class="text-gray-600 pl-1">Titre</span>
5       <input id="getTitle" type="text" name="titre" class="" autocomplete="text"
6         placeholder="modifier le titre" />
7     </label>
8
9     <label class="block w-full">
10      <span class="text-gray-600 pl-1">Description</span>
11      <input id="getDescription" type="text" name="description" class="" autocomplete="text"
12        placeholder="modifier le titre" />
13      <input id="idForm" class="hidden" name="id">
14    </label>
15
16    <label class="block w-full">
17      <span class="text-gray-600 pl-1">Catégorie</span>
18      <select id="getCategory" name="categorie">
19        {{#each listCategories}}
20          <option value="{{id}}">{{name}}</option>
21        {{/each}}
22      </select>
23    </label>
24  </div>
25  <!-- Modal footer -->
26  <div class="flex items-center p-6 space-x-2 rounded-b border-t border-gray-200">
27    <button data-modal-toggle="edit-modal" type="submit"
28      class="text-white bg-blue-700 hover:bg-blue-800 focus:ring-4
29        focus:ring-blue-300 font-medium rounded-lg text-sm px-5 py-2.5 text-center">Enregistrer</button>
30    <button data-modal-toggle="edit-modal" type="button"
31      class="text-gray-500 bg-white hover:bg-gray-100 focus:ring-4
32        focus:ring-gray-300 rounded-lg border border-gray-200 text-sm font-medium
33        px-5 py-2.5 hover:text-gray-900 focus:z-10">Annuler</button>
34  </div>
35 </form>

```

J'ai utilisé deux modules appelés **multer** et **sharp** pour gérer les images. Avec le module **multer**, les membres peuvent envoyer leurs images. Le module **sharp** nous permet de modifier les images. Il est aussi utilisé pour créer une image au petit format.

Les images sont stockées dans le dossier « public » et son sous-dossier « upload ». Les noms des images vont être changés avant d'être stocké sur ce dossier pour qu'il n'y est pas de doublon.

J'ai utilisé la fonction **JOIN** dans MySQL pour créer des relations entre les tables.

```
1 commentSelectAllByPhoto: async (id) => {  
2   return await query(  
3     'SELECT comment.text, membre.name FROM comment INNER JOIN  
4       membre ON comment.membre_id = membre.id WHERE comment.photo_id = ${id}'  
5   );  
6 },
```

Par exemple, dans l'image ci-dessus, dans la commande **query**, j'ai pu obtenir le nom du membre qui a laissé le commentaire en utilisant la **forign_key membre_id**.

Le résultat de cette commande de requête est sous la forme d'un tableau.

Dans la première colonne, on affiche le commentaire et dans la seconde colonne, on affiche le nom de son auteur :

text	name
nice	admin

Pour obtenir les commentaires envoyés à partir d'une image spécifique, j'ai utilisé la **forign_key photo_id** dans la table « comment ».

VIII.5. Conclusion

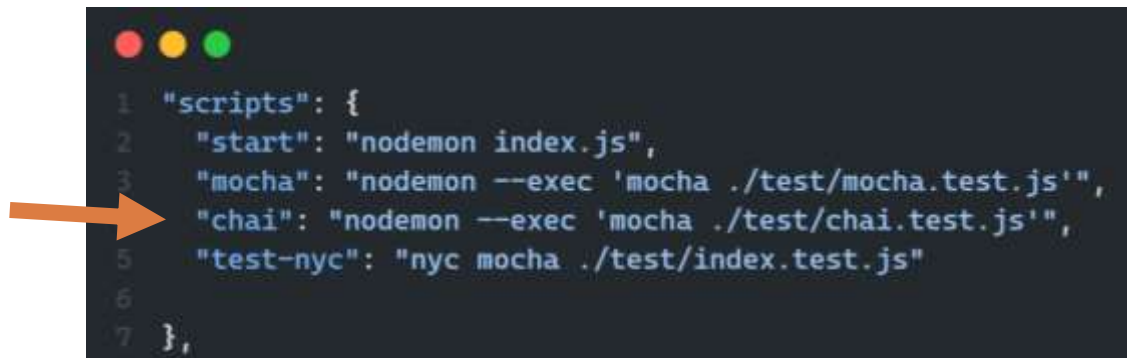
Cette fonctionnalité couvre toutes les compétences que j'ai acquises durant cette formation, résumées par l'utilisation de : NodeJS, JavaScript (JSDOM), MySQL, les modules NPMs (multer, sharp).

Les tests unitaires ont été réalisés avec Mocha et chai sur mon CRUD photos lors de la création de cette fonctionnalité



IX. TEST UNITAIRE

J'ai effectué mes tests unitaires avec les modules Mocha et Chai sur le CRUD des images.



```

1  "scripts": {
2    "start": "nodemon index.js",
3    "mocha": "nodemon --exec 'mocha ./test/mocha.test.js'",
4    "chai": "nodemon --exec 'mocha ./test/chai.test.js'",
5    "test-nyc": "nyc mocha ./test/index.test.js"
6  },
7

```

Test unitaire avec Mocha pour ajouter une image :

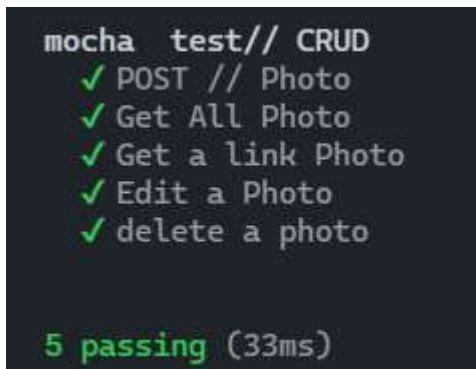


```

1  // CREATE PHOTO
2  it('POST // Photo', async () => {
3    const photo = await photoCreate(data, data.thumbnail, data.photo, data.membre_id)
4    assert(photo)
5    const photoID = await photoSelectAllById(photo.insertId)
6    image.id= photoID[0].id;
7    assert.strictEqual(photoID[0].title, data.title)
8    assert.strictEqual(photoID[0].description, data.description)
9    assert.strictEqual(photoID[0].cat_id, data.categorie)
10   assert.strictEqual(photoID[0].link, data.photo)
11   assert.strictEqual(photoID[0].thumbnail, data.thumbnail)
12   assert.strictEqual(photoID[0].membre_id, data.membre_id)
13   assert.notEqual(photoID[0].alt, data.alt)
14
15 })

```

Tests effectués avec Mocha :



```

mocha test// CRUD
  ✓ POST // Photo
  ✓ Get All Photo
  ✓ Get a link Photo
  ✓ Edit a Photo
  ✓ delete a photo

5 passing (33ms)

```

POST : ajouter une image

GET ALL : récupérer toutes les images

GET A LINK : récupérer le lien d'une image

EDIT : modifier une image

DELETE : supprimer une image

Test unitaire avec Chai pour tester l'API :

```

1  it(' Chai Router // Get a photo', (done) => {
2    chai.request(app)
3      .get(`/api/galerie/${photo.id}`)
4      .set('Accept', 'application/json')
5      .end((err, res) => {
6        if (err) return done(err)
7        res.should.have.status(200);
8        res.should.be.a('object');
9        done();
10     });
11  });
12

```

Tests effectués avec Chai :

```

✓ Should Get the photo
✓ Should Get All Photos from database
✓ should add a new photo to database (58ms)
✓ should edit a photo from database
✓ should delete a photo from database
✓ should delete a photo from database

```

POST : Test de la route pour ajouter une image

GET ALL : Test de la route pour récupérer toutes les images

GET A LINK : Test de la route pour récupérer une image

PUT : Test de la route pour modifier les détails d'une image

DELETE : Test de la route pour supprimer une image

X. RECHERCHES EFFECTUÉES EN ANGLAIS

L'un des problèmes que j'ai rencontrés lors de la création de ce projet, était de passer les données à un modal.

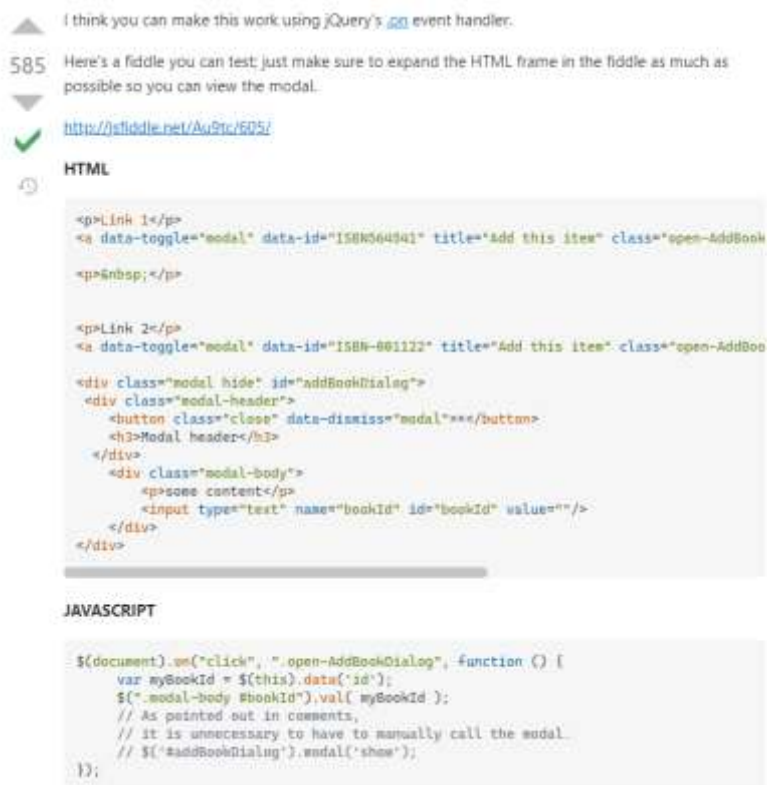
La première solution consistait à créer un modal pour chaque élément et mettre les données de ces éléments dans ce modal lors du rafraîchissement de la page. Mais cela n'avait aucun sens et je cherchais une meilleure solution pour éviter d'écrire du code en doublon.

Donc, après avoir effectué des recherches, j'ai trouvé une réponse, mais pour utiliser cette seconde solution, j'aurai dû utiliser la librairie jQuery.

Faute de temps, j'ai dû choisir cette seconde solution. Elle consistait à placer les données que je voulais passer sur le modal, en tant qu'**attribut** avec l'extension de data sur chaque élément.

En cliquant sur un élément de l'événement et à chaque clic, les informations avec l'extension de data de l'élément ont été stockées dans une variable.

J'ai ensuite utilisé l'id pour transférer ces informations sur le modal.



<https://stackoverflow.com/questions/10626885/passing-data-to-a-bootstrap-modal>

XI. VEILLE EFFECTUÉE SUR LES VULNÉRABILITÉS DE SÉCURITÉ

L'une des parties les plus importantes de la création d'un site ou d'une application est de prêter attention aux problèmes de sécurité. Il est impossible d'atteindre une sécurité à 100 %, même les grandes entreprises rencontrent parfois des bogues de sécurité. Mais en prenant certaines mesures, le site peut toutefois être optimisé en sécurité.

Les mesures que j'ai prises pour la sécurité de ce projet peuvent être divisées en 2 parties.

Lors du développement du site :

- stocker les informations sensibles dans un fichier « **.env** » séparé
- enregistrer les données de connexion des utilisateurs avec des sessions express dans la base de données
- hacher les mots de passe des utilisateurs avec le module **bcrypt**

Sur le serveur :

- activer le pare-feu
- créer un SSL par **let's encrypt** et le module **cerbot**
- désactivation du port sensible 22 et utilisation d'un autre port

XII. CONCLUSION

La formation était très courte et très intense. Il y avait beaucoup de choses à apprendre et beaucoup d'exercices pratiques, ce qui est primordial pour comprendre et s'améliorer. J'ai appris des méthodes qui permettent de bien s'organiser, pour éviter des problèmes.

Avant cette formation, j'étais autodidacte et très curieux de l'environnement du développement. Aujourd'hui, depuis cette formation, je suis encore plus motivé dans mes objectifs professionnels, je suis convaincu de vouloir faire ce métier, qui est devenue une véritable passion pour moi.

J'ai travaillé dans un environnement convivial et bienveillant. J'ai travaillé au sein d'un groupe gentil et solidaire et accompagné par un formateur très gentil, patient et intelligent.